Expansion-based Service Workflow Replanning with Limited Change

Jian Feng Zhang, Ryszard Kowalczyk and Boris B. Wu
Faculty of Information and Communication Technology
Swinburne University of Technology
Hawthorn, VIC 3122, AUSTRALIA
email:{jfzhang;rkowalczyk;bwu}@ict.swin.edu.au
http://www.it.swin.edu.au/centres/ciamas/

Abstract: Service Oriented Architecture and Web Services provide an integration infrastructure for service workflow composition and execution. Many planning techniques can be facilitated to construct the service workflow at a building phase (planning), however, it is difficult and challenging to maintain the qualities of service workflow at execution phase, as the delayed response time, the limited throughput and service faults are common that often violate, or fail the service goals. Then service workflow re-planning should be in place in order to deal with the unpredictable situations during service workflow execution. In this paper, we consider an expansion-based replanning algorithm with the use of a limited change strategy. It addresses the problem of least affecting the original service workflow structure whiles the service goals of the workflow remains valid. We will show simulation results of the proposed replanning strategy based on experiments with service workflow replanning. The results of the simulation are analyzed for the consideration of further research.

1 Introduction

Web service (WS) and service oriented architecture (SOA) have received a wide interest, as their underlying infrastructure enables loosely-coupled integration and composition of services. Composing a number of services can be regarded as the WS counterpart of workflow planning, where the service activities, for example service invocation activities, replace the workflow task activities while the control activities remains same according to business process logics. Then the composed services for a given service goal can be executed as a workflow automatically. However, WS environments are complex, uncertain and dynamic, and service workflows do not always proceed as planned. Hence, one of the challenges is to respond to the changing situations timely and properly during service workflow execution.

Traditionally in *plan-then-execute* life cycle, it relies on anticipating possible situations in the planning phase. Exception handling techniques are utilized to accommodate various situations occurring in execution phase. This class of approaches have commonly been applied to adaptive workflow systems [18][13][14]. A typical example is illustrated in [4], where preassembled repair plans are prepared in advance, and are invoked to deal with spe-

cific exceptions during worflow execution. Those approaches have possible exception to be enumerated in advance, which requires substantial work of modeling and identification during the planning phase.

For a planned workflow to be executed in dynamic and uncertain environments such as web services, it is hard to anticipate possible exceptions as well as to prepare handling means. Service failures are unpredictable. Even if a service can be identified in advance to likely fail, it is hard to foresee what services will be available for replacement when the failure actually happens. While plan-then-execute approaches lack flexibility needed by service workflow, recent approaches, so called *monitoring-and-replanning*, have shown some advantages. The basic idea of replanning is to generate a new plan in case when one or more services have problems during execution [10]. It can be seen as a process that removes the services which possibly hinder the reachability of the goal, and adds new services in order to reach the goal [15]. In practice, replanning should take the service costs into account in addition to resolving the failure. We argue that replanning strategies would be an important factor that affects the quality of replanning, and different concerns for the costs may influence the preferences for replanning strategies.

Motivated by previous work in replanning, we propose an approach to adapt service workflow plans during execution. It is tailored from multi-agents based Adaptive Service Level and Process Management(ASAPM) project currently being carried out in our research center. In this approach agents monitor the execution progress. Each time a faulty service is detected, the replanning agent is triggered to launch the replanning process. The replanning is based on an expansion-based mechanism with the principle of limited change to the original plan, which will be presented in this paper together with the experimental results and analysis.

The rest of this paper is organized as follows. In Section 2 we briefly introduce the background of this research, including assumptions and aims. Section 3 presents our approach in more details in terms of the architecture and algorithms. In Section 4, a scenario is studied as an example to explain the approach. Then we show some experimental results in Section 5, and finally discussion and observation of the approach are given to conclude this paper.

2 Problem Statement and Requirement Analysis

This paper complies with traditional interpretation of workflow replanning, i.e. replanning is triggered during the service workflow execution, and the triggering events could be either functional failures of services, or violations to the specified Quality of Service (QoS). Figure 1 shows how the replanning works. The service pool is the source of services, which is assumed to provide adequate facilities, e.g. indexing and searching, for the planner to find services with required capabilities. As we mentioned in the previous section, replanning strategy is an important factor that makes one replanning different from others. In this section, we discuss some important considerations that we were concerned with in architecture design and strategy development.

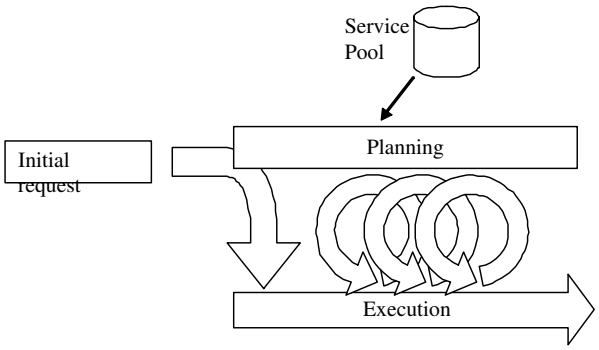


Abbildung 1: The concept of replanning at execution

2.0.1 Replanning costs

There are two types of costs involved in service workflow planning. One is referred to as the service cost, e.g. the total execution cost or the number of services in a planned workflow. The service cost is usually one of the QoS attributes, and is commonly employed to evaluated the quality of a plan. The less is the total service cost, the better is the plan. In the context of replanning, service cost can be used to measure the replanning efficiency, e.g. a good replanning preserves a plan's service costs while maintaining its reachability to the goal. However, the service cost is not the only criterion in replanning. There is another type of cost which is referred to as replanning cost, e.g. the time spent in replanning process. The replanning cost might not necessarily be introduced to service users, but it is an important criterion for the evaluation of a service management system. The less is the replanning cost, the better is the replanning performance. In the development of a replanning system, both costs need to be taken into account. Heuristically, limiting the changes to the old plan reduces the overall cost.

2.0.2 Service information abstraction

Web service provides a high level of information abstraction. Services expose their information through standard interface in standardized syntax/semantics such like WSDL, and hide the details of internal mechanisms and operation behaviors. On the one hand, it facilitates many automated processing, including automated composition. On the other hand, the planner can only see the services as "black boxes". Hence some techniques succeeding in traditional workflow systems are not suitable for WS workflow due to the lack of the knowledge of internal mechanisms. For example, failure of a truck delivery service may be caused by flatted tyre or engine problems. In the approach presented by [4], the planner may add a task of replacing the flat tyre or fixing the engine to resume the delivery. In the service environments, this is not the case since the information of flatted tyre or engine problems of the service will not be available at the time of replanning. The planner will remove the service in whole instead of fixing the service. Replanning in Web service

workflow can rely only on the abstracted service information exposed through standard interface.

2.0.3 Dynamic service environment

Web service environment is uncertain and dynamic. Errors arise unpredictably. Situations, such as the availability of services, keep changing. Exception-handling alike approaches, including those using prepared repair plans, are not suitable for such a dynamic environment since it is impossible to identify all, if any, possible failures and prepare recovery schema in advance. Furthermore, since the services may become available/unavailable at any time, the repairing plans prepared in advance could become obsolete.

The replanning architecture and strategy that we are going to described in the next sections are based on the analysis of problems and requirements given above, designed to meet the needs of cost-effectiveness, service orientation and flexibility. It is referred to as *expansion-based replanning* in the following sections. And its major features can be described as:

- Limited changes to the old plan so that the overall cost can be kept at minimal level.
- No reliance on anticipation since the replanning is invoked during service execution and takes the advantages of service information abstraction.
- Fully automated so that no human intervention is needed during replanning process, and the system can make timely response to exceptions in execution.

The system works by "monitoring-and-replanning". When a service request from a service user arrives, the planner will composes an initial service workflow plan that satisfies the user's functional and non-functional requirements. Then the workflow is enacted and executed, and this phase is referred to as workflow execution. Execution progress is monitored through till the end of this phase. Whenever service errors and violations occur during execution, the replanning process is invoked to fix the workflow plan. During replanning, an expansion-based mechanism is utilized to limit the changes to the old plan.

3 The Architecture and Strategy

In this section, we introduce the replanning architecture and the replanning strategy. The architecture of the replanning is developed based on the multi-agents technologies, where each service has an agent, referred to as a service agent, whose tasks are to manage the service, e.g. service invocation and monitoring, as well as interact with other service agents. Not like the service agents, a planner agent will not manage individual services directly. Instead, it interacts with service agents to collect information necessary for replanning, e.g., the states of the services involved in replanning, and the "neighborhood" of the service agents.

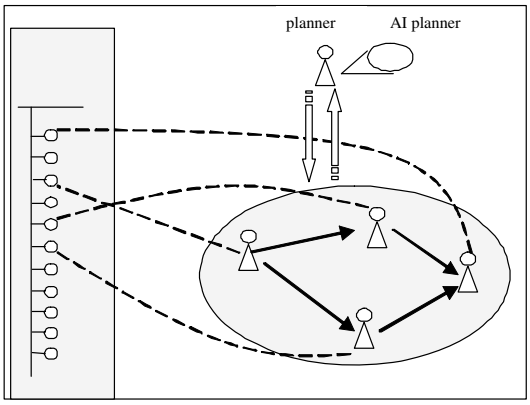


Abbildung 2: the multi-agent based replanning architecture

3.1 The replanning architecture using Multi-Agents

Figure 2 shows a graphic view of the multi-agents based architecture of service workflow replanning. We assume that a service pool is available (in the context of SOA, the service pool is regarded as a service database such like UDDI). The planner (planner agent) has an access to the service pool in order to search for particular services and capabilities.

Current Web services are stateless entities. The previous work seldom consider the state of the services in replanning. In our approach, a service agent is created for each service to manage it, whose job includes monitoring the local state of the service and managing the connections to its "neighborhood". In such a way, a stateless service becomes a stateful service. The planner agent consults with the service agents about service states during replanning. From workflow execution point of view, the multi-agent workflow is executed in distributed manner. The service agents and the planner agent are created on JADE platform. The protocol used for communications among agents is ACL - agent communication language. While the interaction between services and corresponding service agents is implemented in a proprietary way, it is easy to be adopted to WS standards(WSDL/SOAP), via WS2JADE [17], for example, which provides an integration mechanism for agent system and Web service.

When an initial request comes, the planner selects services from the pool to compose a workflow plan. Each service is managed by a service agent. When service agent detects its managed service is not available due to failures or availability, it reports to the planner. The planner replans to repair current plan, then the execution continues. We use an existing AI planning engine to make initial plan and replan. Replanning is carried out by iterations of ëxpanding problem area and trying", i.e. expansion-based replanning, which is to be described in the next section. In each iteration, the planning engine gets a planning problem constructed by the planner agent as its input, and outputs a plan of the problem or notice of failure. In another word, it is not needed to modify the codes or algorithms of the planning engine being utilized in this strategy.

3.2 Expansion mechanisms for replanning

To limiting the changes to the old plan, we consider an expansion-based mechanism that is utilized in replanning discussed in the following subsection. Our observation on replanning processes tells that the less services are involved in replanning, the less services are probably changed before a valid new plan is reached. If a service in the current plan fails, our approach starts by considering the failed service as a replanning area, and looking for a service or an equivalent subplan to replace the failed service. In the case that there is not a replacement service or equivalent subplan available, we expand the replanning area around the failed service, so that it also covers a number of services adjacent to the failed one. Then we look for a service or an equivalent subplan to replace the enlarged area. If such a replacement is found, some existent services are removed and some new services are added. If no replacement is found in this area, then larger and larger areas are considered.

From the viewpoint of planning, we can describe the expansion mechanism as follows: Given a set of initial states I and a set of goal states G, a plan is such a network of services that the services are connected by matching preconditions with effects, with the preconditions unmatched within the network being satisfied by I, and the effects unmatched within the network satisfying G. If a service fails, we define a replanning area, which includes only the failed service initially. The services adjacent to the replanning area have their preconditions and/or effects unmatched. Let the set of unmatched effects be I', and the set of unmatched preconditions be G', the problem of fixing the failure can be seen as the problem of composing a valid plan for the replanning area, with I' as initial states and G' as goal states.

If no plan can be found with I' and G', the replanning area is expanded and covers some services previously adjacent to the boarder. A new set of initial states Iänd a new set of goal states Gäre derived and assigned to I' and G'. The new sets of I' and G' are ready for another try of planning. In this way, the process goes on until the plan is fixed or there is no place allowing for expansion.

A replanning process consists of a number of iterations. Each of the iterations consists of three basic operations, constructing the planning problem, searching for a sub-plan and expanding the replanning area.

- *Constructing planning problem* : The planner constructs I' and G' by consulting the service agents adjacent to the replanning area: what conditions are available and what conditions need to be satisfied.
- *Searching for a plan* : The planner searches the service pool, looking for proper services to compose a subplan for the current problem.
- *Expanding replanning area* : If the planner fails to find such a subplan, planner removes some of the services on border of the replanning area so that the planner can exploit chances with new I' and G'.

If there exists a solution, in the worst case a plan will be found when the replanning area

has been expanded as large as the original plan. However, in most cases, when a large number of services are available, replanning is expected to succeed in a smaller area.

3.3 Expansion Strategies

The framework is not bound to specific strategies of expanding the replanning area. Here we present some strategies based on the plan's structure and the services' status and properties.

(a) We can expand the replanning area blindly forward and backward alternatively, thus it is expanded with the failed service as the central point. While it is the simplest one among the strategies tested in our experiments, it performs as effective as others.

(b) Considering that when the replanning is launched, the services before the replanning area (near the initial states) are more likely finished already or in execution, we can expand the replanning area forward (to the goal) first to look for a solution, and expand backward (to the initial) if no solution is found after the expansion reaches the goal.

With the same observation but different concerns, considering that it is more complex to stop an executing service or undo a finished one than removing a service not executed yet, we can expand the replanning area backward first so that planner can try replanning before more services start or finish execution.

(c) We also propose an expansion strategy guided by a heuristic motivated by A* search [8]. This heuristic is based on expected cost of finding a new plan after removing a node. Among the nodes on the boarder of the replanning area, the one with the lowest heuristic value is chosen to be removed.

The heuristic value of node n is calculated as f(n) = g(n) + h(n), where g(n) is the distance from the failure point to n, and h(n) is the number of conditions that will become unsatisfied if node n is removed.

The idea behind g(n) is that the resulted plan is expected to be no better than original one, which means the distance from the failed node to n in the resulted plan is no shorter than the distance in the original plan. The idea behind h(n) is that the number of unsatisfied conditions suggests how difficult it is to find a sub-plan. This heuristic is based on "localänd "current"data, i.e., the planner can calculate it by consulting only agents involved in current replanning till now.

In the above strategies we did not consider the services' execution status. When replanning is launched, the plan is partially executed, some services have already finished, some are executing and others are not executed yet. Services of different execution states may have different preference to be removed. We model a service's execution states as finished, executing and not-executed-yet, and a service has two parameters indicating whether it can be undone after execution and whether it can be stopped in execution. The service agents

report such information when they are consulted by the planner. We combine following priority rules with above strategies:

1. As for a finished service, if it can not be undone, then it can not be removed; if it can be undone, then it can be removed;
2. As for an executing service, if it can not be stopped and undone safely, then it can not be removed; if it can be stopped and undone safely, then it can be removed;
3. As for a not-executed-yet service, it can be removed;
4. Not-executed-yet services have higher priority to be removed than executing services;
5. Executing services have higher priority to be removed than finished services.

Thus the services fall into three catalogues: can not be removed, prefer not to be removed, and can be removed. Similarly, a priority queue can be constructed according to services' utilities/costs. We describe the strategies here to show that the framework allows strategies based on various concerns and preferences. Further discussion is out of the scope of this paper.

4 A Scenario

We present a scenario in this section to illustrate how expansion-based replanning works in two situations: a failure happens to a service which is being executed, and a service is found to be unavailable before it starts to execute.

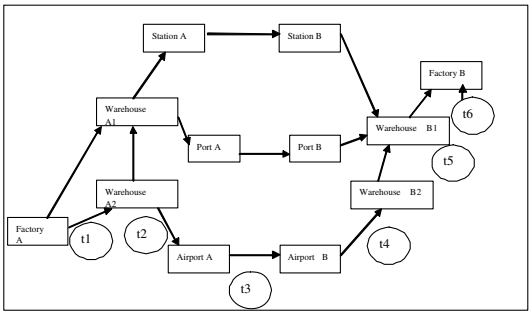


Abbildung 3: The workflow of transportation

We suppose a company needs to transport a quantity of goods from factory A to factory B. The map (Fig 3) shows there are ports, airports, train stations, etc between the two factories. A number of delivery services are available to be hired (arrowed lines between the sites in Fig 3). We choose a number of delivery services and compose them into a plan that is indicated as the dash line in Fig 3 where

t1: Factory A to Warehouse A2

t2: Warehouse A2 to Airport A

t3: Airport A to Airport B

t4: Airport B to Warehouse B2

t5: Warehouse B2 to Warehouse B1

t6: Warehouse B1 to Factory B.

Each of the tasks is managed by a service agent, in the order of the task flow of the composed services:

$$t1 \rightarrow t2 \rightarrow t3 \rightarrow t4 \rightarrow t5 \rightarrow t6 \quad (1)$$

Now let us consider two situations to see how the replanning is carried out.

Situation 1 - Service fails in execution

Current status : task t1 is finished; Task t2 is in execution; tasks t3-t6 are waiting for execution; i.e. the goods is on the way from Warehouse A2 to airport A.

Error : t2 fails to execute and its agent reports the failure to the planner.

Replanning process is shown in Figure 4 :

1. Planner looks for a subplan to replace t2, i.e. transporting goods from Warehouse A2 to Airport A, and fails.
2. Planner expands the replanning area. Since t1 has already finished, it can not be removed. Planner expands the replanning area forward, removing t3.
3. Planner looks for a sub-plan that transports goods from WarehouseA2 to AirportB, and fails.
4. Planner expands the replanning area. Again, it can only be expanded forward, and t4 is removed. Planner looks for a subplan that transports goods from WarehouseA2 to WarehouseB2, and fails.
5. Planner expands the replanning area forward, removing t5. It looks for a subplan that transports goods from WarehouseA2 to WarehouseB1. This time it finds a subplan successfully: a) t7: WarehouseA2'WarehouseA1 b) t8: WarehouseA1'PortA c) t9: PortA'PortB d) t10: PortB'WarehouseB1
6. Replace t2-t5 in original plan with t7-t10, and the plan is fixed.

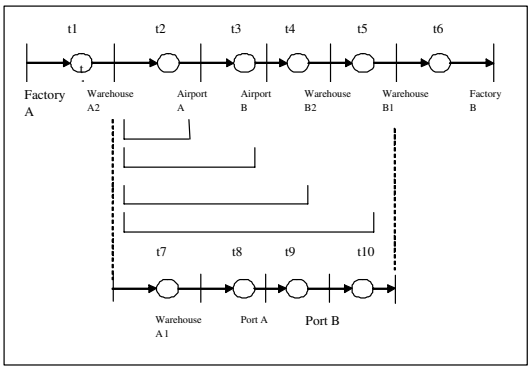


Abbildung 4: Situation1

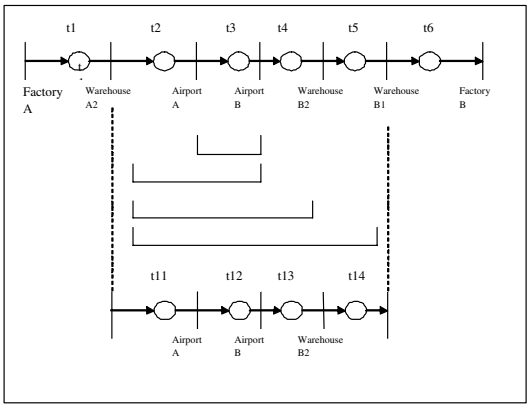


Abbildung 5: Situation 2

Situation 2 - Service becomes unavailable before being executed

Current status : same as situation1,i.e. t1 is finished; t2 is in execution; t3-t6 are waiting for execution.

Error : The goods is on the way from Warehouse A2 to airport A. Then the agent monitoring t3 (from Airport A to Airport B) finds the scheduled flight is canceled for bad weather.

Replanning process is shown in Figure 5

1. Planner can not find transport from Airport A to Airport B. It removes task t2 from plan. (Task t2 is in execution. We suppose the agent in charge of t2 agrees to cancel t2.)

2. Planner can not find transport from Warehouse A2 to Airport B. It removes task t4 from plan.

3. Planner can not find transport from Warehouse A2 to Warehouse B2. Task t1 has already finished, and we suppose the agent in charge of t1 refuses to cancel t1. Task t5 is removed.

4. Planner finds a way to transport goods from Warehouse A2 to Warehouse B1. The plan is fixed.

5 Experiment Result

For the experimental validation of our approach, we implemented a prototype system with Jade as agent platform and FFv1.0 [9] as AI planning engine. In our framework we use AI planning engine as a black-box, not depending on specific implementation techniques. FF suits this usage well since it supports standardized input format (PDDL).We implemented a prototype with Jade agent platform and FFv1.0 planner for experiment and illustration. Fig 6 shows its GUI.

Abbildung 6: GUI of the replanning system

The service model in our experiment is simple but enough for testing our approach. A service has a set of preconditions and a set of effects. Preconditions and effects are presented as the existence of sets of tokens. We devised service domains for test. (5 domains, each of which has 2000 randomly generated services.) We compared the number of changed services in replanning from scratch with the number in expansion-based replanning. In expansion-based replanning we used 3 of the expansion strategies mentioned before. In the experiment, the system triggers replanning by choosing a service randomly in a plan and stopping it to simulate a failure. In fig 7 we can see that the planner fixes each failure in 4 ways to allow their comparison: from scratch, expanding forward and backward alternatively, expanding forward till end then backward, expanding by checking heuristic values. In general, the results show that expansion-based replanning removes and adds less services than replanning from scratch, and the performance of the 3 expansion strategies

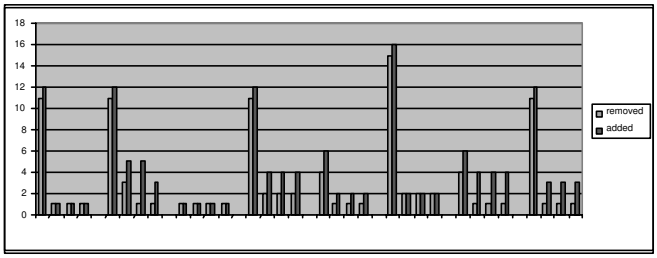
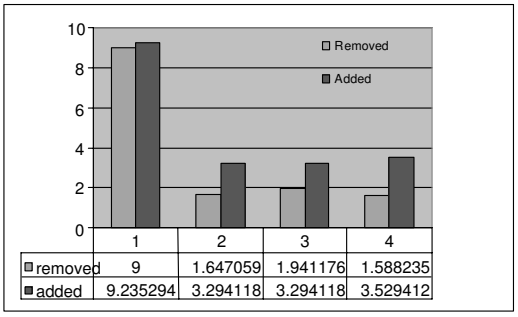


Abbildung 7: Numbers of removed and added services in replanning on a plan (Experimental results of replanning on 8 failures happening in a plan are shown in this chart. Each failure is fixed in 4 ways (left to right in each group of columns): 1: from scratch 2: expanding backward and forward alternatively 3: expanding forward till end then backward 4: expanding by checking heuristic value.)

are similar in most cases. More experiments show the same result (fig 8) except for a small number of exceptional cases.



	1	2	3	4
removed	9	1.647059	1.941176	1.588235
added	9.235294	3.294118	3.294118	3.529412

Abbildung 8: Average numbers of removed and added services (1: from scratch 2: expanding backward and forward alternatively 3: expanding forward till end then backward 4: expanding by checking heuristic value)

Another observation is that the number of added services and the number of removed services are in an approximate direct ratio: the number of added services is no less than, and will not be too much larger than the number of removed services, which we can see more clearly in fig 9. The former fact is easier to interpret since a good planner makes an optimal plan at first and has to make a less good one after a useful service fails. The reason for the latter fact is that: if there are services available in the pool, which are similar to the removed ones, we can expect the new plan is similar to the previous one, though we can not predicate the level of similarity exactly. This observation suggests that we are probably able to, to some extent, control the number of added services by controlling the number of removed services, and the number of removed services can be controlled by tuning the size of the replanning area. In other words, the size of the replanning area is hopefully an effective means of adjusting the range of cost ready to pay for changing running plans, thus the balance of changing cost and resulted plan's optimality can be achieved by tuning

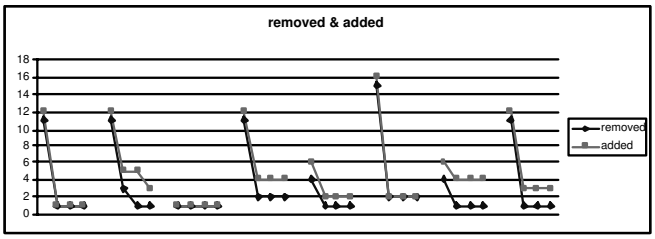


Abbildung 9: Numbers of removed and add services

this parameter according to business requirements.

6 Related Work

Brian Drabble [4] presented plan-repair mechanisms similar to the idea of exception handling in programming languages. When a condition that deviates from the normal flow of execution occurs, the execution switches to a predefined repair plan. Repair plans deal with specific failures. After the repair plan is completed, the execution switches back to the normal flow. That approach can not deal with failures that are not predicted in planning phase. And if it is in dynamic WS environment, the prepared repair plans may become outdated before they are invoked. Our approach does not require subplans to be prepared in advance.

GPG [1] is a domain independent planner capable of planning from scratch and replanning in case the plan becomes invalid due to changed original initial states or goal states. Both planning and replanning are based on graph planning [2]. When a plan becomes invalid, GPG constructs a new planning graph for altered initial and goal states. It looks for an adapted plan with two search techniques. The systematic search technique uses the new planning graph to compare with the old plan in order to find out inconsistencies. The local search uses the new planning graph together with the old plan to construct a partial plan, from which the algorithm searches through the plan space. GPG deals with the failures arising from the changed initial and goal states. In our approached we deal with both the failures within the plan and on the initial and goal states.

Replan (Boella and Damiano 2002) organizes actions (action library) in an abstraction hierarchy similar to hierarchical task network, where an abstract action can be decomposed into a sequence of primitive or abstract actions. The planning process starts from decomposing the topmost action in the hierarchy that achieve the goal. Decomposition goes on until it obtains a set of plans. A partial plan is sequence of abstract and primitive actions that achieve the goal, and a plan contains only primitive actions. Each plan is associated with a derivation tree, which is built during the planning process and will be used in replanning. The tree shows how the primitive actions in the plan are derived from abstract

actions. In replanning, the planner locates an invalidated leaf node, removes the subtree containing the node, and refines the tree (partial plan). If the refinement fails, it removes larger subtree that contains the failed node and starts refinement again. That approach requires the history of planning(derivation tree) is recorded and accessible in replanning. Our approach does not require records of the history data.

deWeerdt et al [6][7] and Krogt [15] presented a general view of replanning/plan repair by modifying a general planning model called refinement planning [10]. Refinement planning sees the construction of a plan as an iterative refinement of the set of all possible plans. deWeerdt [6][7] and Krogt [15] added an operation of unrefinement to refinement planning so that it can model plan repair in general. In the general model, plan repair is a process consisting of two operations: unrefinement, i.e. removing actions from a plan, and refinement, i.e. adding actions to a plan.

As it was remarked by Kambhampati [10], past replanning systems often assume the failure of a plan comes from altered initial or goal states, because they stem from the research on plan reuse: reuse the current plan to solve a new problem that has new initial and goal states. GPG states the assumption explicitly. deWeerdt and Krogt's general framework does not make this assumption explicitly [6][15][7]. However, its algorithm starts unrefinement by removing actions depending on initial states or producing goals, and the plan is shrunk from the outside in. We may consider it an implicit assumption in their work that failures are caused by changed initial and goal states. Under this assumption, execution failures, which happen within the plan, will not trigger the replanning process properly. Our approach removed this restriction. Wherever a failure happens, the replanning process is launched to solve it, and the unrefinement starts from the service where failure happens. Since the initial and goal states are presented as dummy services, changes to initial and goal states will also be dealt with as well as failures of other services.

7 Conclusion and Further Work

In this paper we propose an approach to adapting service workflow plans in response to service failure during execution. Agents are used to monitor the execution progress and start expansion-based replanning when detecting failures. AI planning technologies are used to automate the replanning process, thus the system avoids human intervention. Replanning is carried out in an ëxpanding problem area and tryingßtyle so as to limit the changes to the old plan when generating a new valid plan. The experimental results show that expansion-based replanning can repair a failed plan with less changes to it in comparison with replanning from scratch. We also find that the number of added services is in an approximate direct ratio to the number of removed services, and the latter is related to the size of the replanning area, which suggests the possibility of controlling the changes to the old plan by tuning the size of the replanning area.

Our future work includes examining other strategies for expansion, and investigating mechanisms for fully decentralized replanning, i.e. enabling service agents to repair the plan in cooperation with each other so as to eliminate the need of a dedicated planning agent.

Literatur

[1] Alfonso Gerevini, I. S. (2000). Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques. the Fifth International Conference on AI Planning Systems (AIPS-00), Menlo Park, CA, AAAI Press.

[2] Avrim L. Blum, M. L. F. (1997). "Fast planning through planning graph analysis.Ärtificial Intelligence 90(1-2): 281 - 300.

[3] Bernhard Nebel, J. K. (1995). "Plan reuse versus plan generation: a theoretical and empirical analysis.Ärtificial Intelligence 76(1-2): 427 - 454.

[4] Brian Drabble, J. D. A. T. (1997). Repairing plans on the fly. Proceedings of the NASA Workshop on Planning and Scheduling for Space, Oxnard CA, USA.

[5] DanWu, E. S., James Hendler, Dana Nau, Bijan Parsia (2003). Automatic Web Services Composition Using SHOP2. International Conference on Automated Planning & Scheduling, ICAPS 2003, Trento, Italy.

[6] deWeerdt, R. v. d. K. a. M. (2004). The two faces of plan repair. Proceedings of the BNAIC (BNAIC-04), pp. 147–154, Groningen, the Netherlands.

[7] deWeerdt, R. v. d. K. a. M. (2005). Plan Repair as an Extension of Planning. Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-05), 2005., Monterey, California, U.S.A.

[8] Hart, P. E., Nilsson, N. J.; Raphael, B. (1968). Ä Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4 (2): pp. 100C107.

[9] Hoffmann, J. (2001). "FF: The Fast-Forward Planning System.ÄI Magazine 22(3): 57-62.

[10] Kambhampati, W. C. a. S. (2005). Replanning: A new perspective. Poster Program, ICAPS 2005, Monterey, California, U.S.A.

[11] M. Pistore, P. B., E. Cusenza, A. Marconi, P. Traverso (2004). WS-GEN: A Tool for the Automated Composition of Semantic Web Services. Proceedings of the International Semantic Web Conference, Hiroshima, Japan, 2004.

[12] Matthias Klusch, A. G., Marcus Schmidt (2005). Semantic Web Service Composition Planning with OWLS-Xplan. Proceedings of the 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, 2005, Arlington VA, USA, AAAI Press.

[13] Peter J. Kammer, G. A. B., Mark Bergman (1998). Requirements for Supporting Dynamic and Adaptive Workflow on the WWW. CSCW 1998 Workshop; Towards Adaptive Workflow Systems, November 1998, Seattle, WA.

[14] Peter J. Kammer, G. A. B., Richard N. Taylor,Mark Bergman (2000). "Techniques for Supporting Dynamic and Adaptive Workflow."Computer Supported Cooperative Work vol. 9(no. 3-4): 269-92.

[15] Roman van der Krogt, M. d. W., Nico Roos,Cees Witteveen (2004). Unrefinement Planning: Extending Refinement Planners with Plan Repair Capabilities. Proceedings of the 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG-04), Cork, Ireland.

[16] Subbarao Kambhampati, C. A. K., Qiang Yang (1995). ”Planning as Refinement Search: A unified framework for comparative analysis of Search Space Size and Performance.Ärtificial Intelligence 76(1-2): 167-238.

[17] Xuan Thang Nguyen,Ryszard Kowalczyk (2005). WS2JADE: Integrating Web Service with Jade Agents. Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005) in conjunction with 4th International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2005), Utrecht, the Netherlands, July 25-29, 2005.

[18] Yanbo Han, A. S., Christoph Bussler (1998). A Taxonomy of Adaptive Workflow Management. ACM CSCW 98 workshop proceedings, 1998, Seattle.